UDACITY

Return to Classroom

DISCUSS ON STUDENT HUB

Image Captioning

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on completing the project!

Your model performs well.

Image captioning requires to recognize the important objects, their attributes and their relationships in an image. To learn more,
here is another resource to develop a image caption architecture as well as this resource.

All the best in future projects!

Files Submitted

The submission includes model.py and the following Jupyter notebooks, where all questions have been answered: **2_Training.ipynb, and** **3_Inference.ipynb.**
Excellent! You have provided all of the files and you have answered the questions.

model.py

The chosen CNN architecture in the `CNNEncoder` class in model.py makes sense as an encoder for the image captioning task.

Great! Your `EncoderCNN` class makes sense. I see you have used the pretrained resnet50.

The chosen RNN architecture in the `RNNDecoder` class in model.py makes sense as a decoder for the image captioning task.

Great! `DecoderRNN` class makes sense. Great use of the drop out layer.

2_Training.ipynb

When using the `get_loader` function in data_loader.py to train the model, most arguments are left at their default values, as outlined in Step 1 of 1_Preliminaries.ipynb. In particular, the submission only (optionally) changes the values of the following arguments: `transform`, `mode`, `batch_size`, `vocab_threshold`, `vocab_from_file`.

Great! You left most arguments at their default except for a few.

```
data_loader = get_loader(transform=transform_train,
                         mode='train',
                         batch_size=batch_size,
                         vocab_threshold=vocab_threshold,
                         vocab_from_file=vocab_from_file)
```

The submission describes the chosen CNN-RNN architecture and details how the hyperparameters were selected.

Excellent! You mentioned based your architecture NIC (Neural Image Caption) paper.

The transform is congruent with the choice of CNN architecture. If the transform has been modified, the submission describes how the transform used to pre-process the training images was selected.

Good! The transform is congruent.

The submission describes how the trainable parameters were selected and has made a well-informed choice when deciding which parameters in the model should be trainable.

Excellent! You answered that you trained the last layer of encoder and trained all of the layers of the decoder.

The submission describes how the optimizer was selected.

Good! You used Adam because it is generally a good optimizer.

The code cell in Step 2 details all code used to train the model from scratch. The output of the code cell shows exactly what is printed when running the code cell. If the submission has amended the code used for training the model, it is well-organized and includes comments.

Good job! The 3 epochs are printed.

3_Inference.ipynb

The transform used to pre-process the test images is congruent with the choice of CNN architecture. It is also consistent with the transform specified in `transform_train` in 2_Training.ipynb.

The implementation of the `sample` method in the `RNNDecoder` class correctly leverages the RNN to generate predicted token indices.

Your generated predicted tokens:

```
example output: [0, 3, 335, 1440, 130, 131, 77, 32, 265, 18, 1, 1, 1, 1, 18, 1, 1, 1
, 1, 1]
```

The `clean_sentence` function passes the test in Step 4. The sentence is reasonably clean, where any `<start>` and `<end>` tokens have been removed.

Good job! You cleaned your sentence.

```
example sentence: A large airplane is parked in the sky .
```

The submission shows two image-caption pairs where the model performed well, and two image-caption pairs where the model did not perform well.

Your example images show some that worked well and some that do not.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START